The MIPS Pipelined Processor

Basic Simulations in ModelSim of the Processor without Forwarding and Hazard Resolution

Due: May 15, 2016

By: Avik Bag, Daniel Schoepflin

Contents

Overview

In the second portion of this project (Part 1: Designing the MIPS Pipeline), the goal was to enhance the single-cycle MIPS processor developed in the first portion and convert the design to a pipelined data path MIPS processor. In doing so, the overall throughput of the processor was greatly increased. Interestingly, since the components are ideal, the performance observed on the waveforms does not appear improved, but should the processor from the first component of the project and the processor from this portion be built, this would almost certainly outperform the first. The design of the pipelined processor was based on the diagram pictured on the following page in Fig. 1.

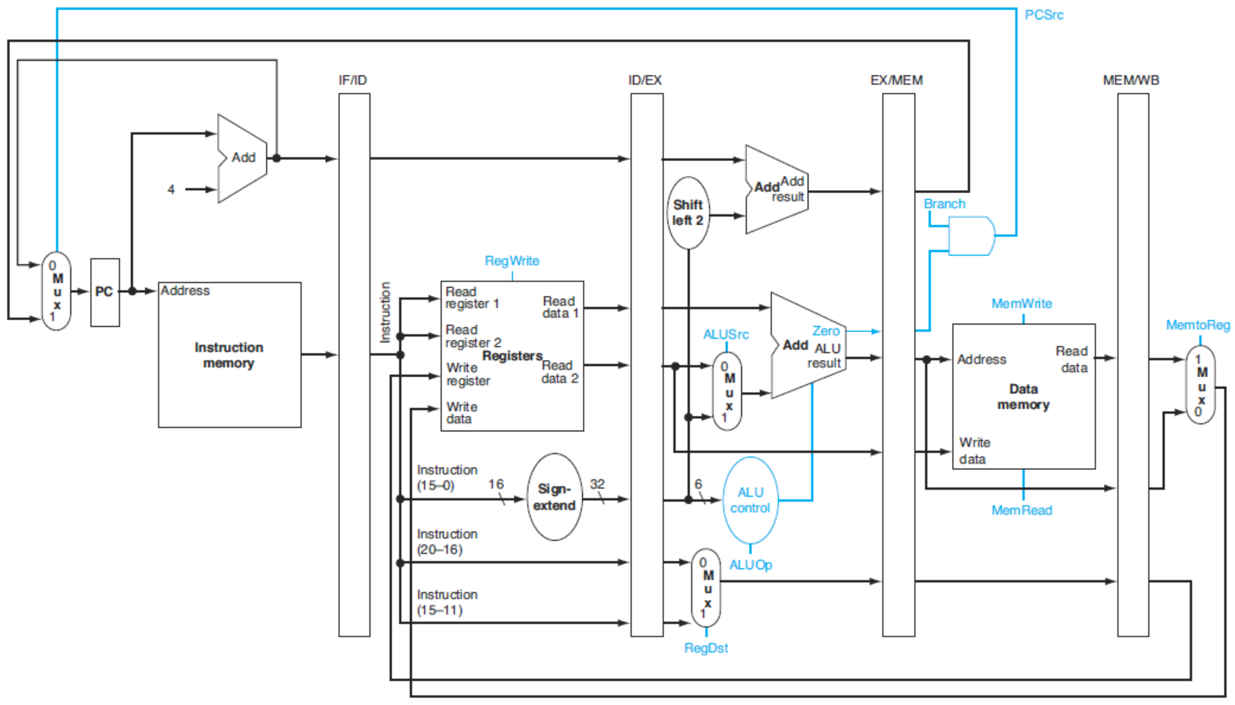


Fig. 1. The pipelined MIPS processor as shown in the Patterson and Hennessy text.

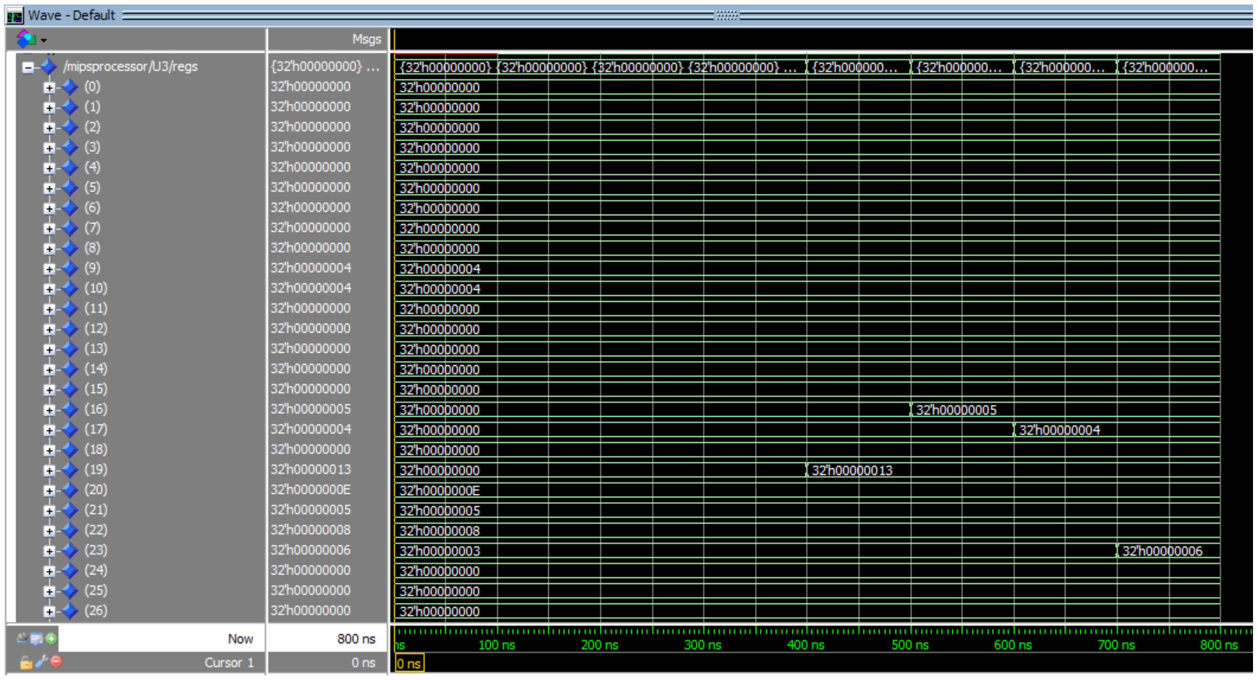
Notice however, that this processor does not resolve hazards. Since this was not required for this portion of the project, this diagram was sufficient. Future improvements to allow for hazard control are discussed in the Further Enhancements and Future Work section of this document. Additionally, the pipeline as shown above does not allow for jump instructions, so that was added in the design of the processor despite not being required for any test components.

Basic Design Principles

When constructing the processor in VHDL, the bulk of the focus of the design was placed on modularity. This was desirable since modularity would allow for the code to be extended and improved upon (as well as easing testing). Additionally, designing the VHDL processor representation as a set of components allowed the branch resolution to be easily adjusted without major changes to the code base as a whole. A second, useful aspect of the modularity was that the design of the code mimicked the diagram of Fig. 1 and suggested the same overall structure. Ultimately, the major change between Part 0 and this part of the project was the addition of the buffers between phases which was accomplished with large registers which stored values with input signals and updated the output signals with this new input on every rising clock edge. In doing so, each was synchronized and every instruction took 5 clock cycles, as desired from the design of the MIPS pipelined processor.

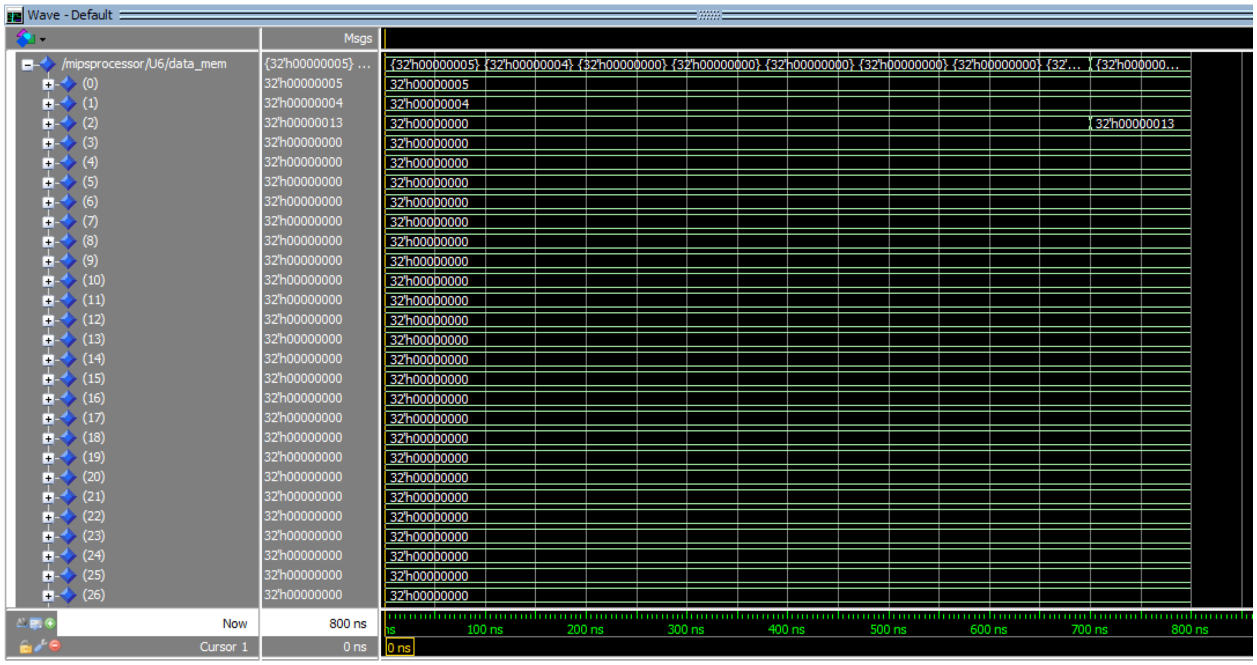
Annotated Result Screenshots and Analysis

In order to test the two designs (that is, the one in which the branch was resolved in the EX stage and the one in which the branch was resolved in the ID stage), sample test code was provided and the processor was simulated in ModelSim. ModelSim facilitated the viewing and observations of the internal signal values which were used to confirm proper functionality of the VHDL code. Figs. 2 and 3 on the following page show the resulting registers and data memory of Program 1, respectively. Since there are no hazards that need to be resolved and no branching, at each clock cycle the PC increments and fetches the next instruction from memory. Every instruction in the pipeline continues to the next stage by using the values stored in the corresponding buffer on the last cycle. Finally, the instruction exits the pipeline in the WB stage.



```
Wave - Default
                              Msgs
/mipsprocessor/U3/regs        {32'h00000000} ...   {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} ... {32'h000000... {32'h000000... {32'h000000... {32'h000000...
  (0)                         32'h00000000         32'h00000000
  (1)                         32'h00000000         32'h00000000
  (2)                         32'h00000000         32'h00000000
  (3)                         32'h00000000         32'h00000000
  (4)                         32'h00000000         32'h00000000
  (5)                         32'h00000000         32'h00000000
  (6)                         32'h00000000         32'h00000000
  (7)                         32'h00000000         32'h00000000
  (8)                         32'h00000000         32'h00000000
  (9)                         32'h00000004         32'h00000004
  (10)                        32'h00000004         32'h00000004
  (11)                        32'h00000000         32'h00000000
  (12)                        32'h00000000         32'h00000000
  (13)                        32'h00000000         32'h00000000
  (14)                        32'h00000000         32'h00000000
  (15)                        32'h00000000         32'h00000000
  (16)                        32'h00000005         32'h00000000                    32'h00000005
  (17)                        32'h00000004         32'h00000000                                  32'h00000004
  (18)                        32'h00000000         32'h00000000
  (19)                        32'h00000013         32'h00000000         32'h00000013
  (20)                        32'h0000000E         32'h0000000E
  (21)                        32'h00000005         32'h00000005
  (22)                        32'h00000008         32'h00000008
  (23)                        32'h00000006         32'h00000003                                                32'h00000006
  (24)                        32'h00000000         32'h00000000
  (25)                        32'h00000000         32'h00000000
  (26)                        32'h00000000         32'h00000000
                    Now       800 ns      ns  100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns
               Cursor 1       0 ns        0 ns
```

Fig. 2. Registers of the MIPS processor for the first program. Note that \$s3 has 19, \$s0 has 5, \$s1 has 4, and \$s7 has 6 by the final clock cycle (clock cycle 8).

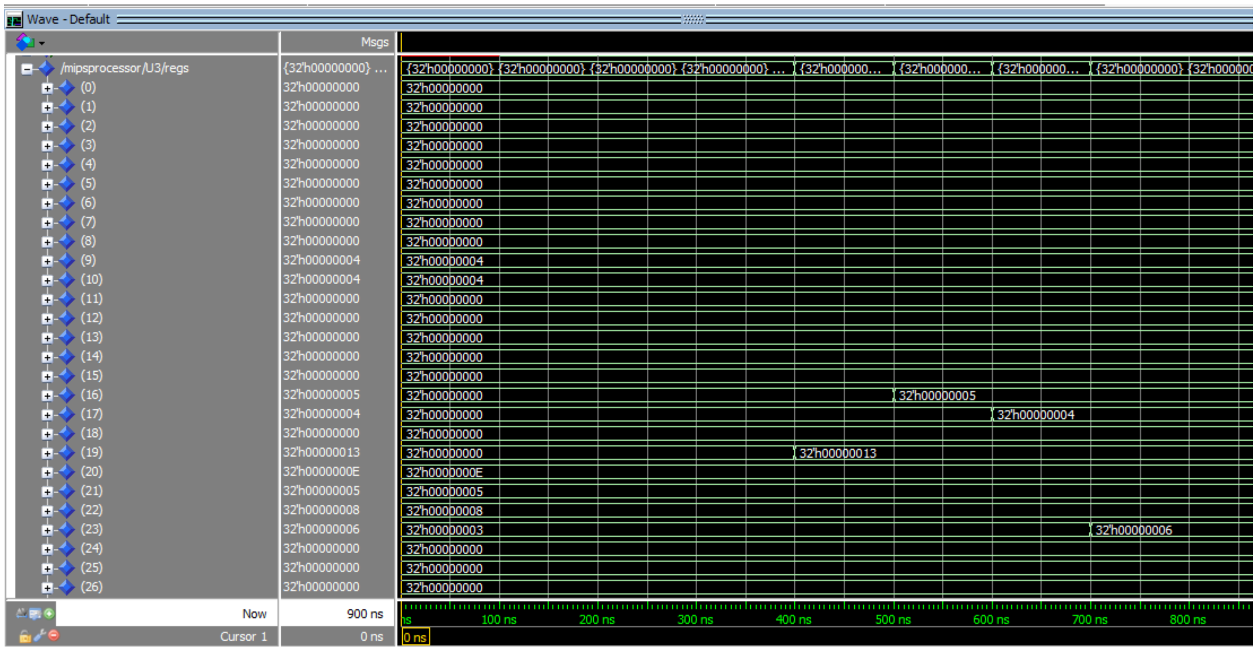


```
Wave - Default
                              Msgs
/mipsprocessor/U6/data_mem    {32'h00000005} ...   {32'h00000005} {32'h00000004} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'... {32'h000000...
  (0)                         32'h00000005         32'h00000005
  (1)                         32'h00000004         32'h00000004
  (2)                         32'h00000013         32'h00000000                                                32'h00000013
  (3)                         32'h00000000         32'h00000000
  (4)                         32'h00000000         32'h00000000
  (5)                         32'h00000000         32'h00000000
  (6)                         32'h00000000         32'h00000000
  (7)                         32'h00000000         32'h00000000
  (8)                         32'h00000000         32'h00000000
  (9)                         32'h00000000         32'h00000000
  (10)                        32'h00000000         32'h00000000
  (11)                        32'h00000000         32'h00000000
  (12)                        32'h00000000         32'h00000000
  (13)                        32'h00000000         32'h00000000
  (14)                        32'h00000000         32'h00000000
  (15)                        32'h00000000         32'h00000000
  (16)                        32'h00000000         32'h00000000
  (17)                        32'h00000000         32'h00000000
  (18)                        32'h00000000         32'h00000000
  (19)                        32'h00000000         32'h00000000
  (20)                        32'h00000000         32'h00000000
  (21)                        32'h00000000         32'h00000000
  (22)                        32'h00000000         32'h00000000
  (23)                        32'h00000000         32'h00000000
  (24)                        32'h00000000         32'h00000000
  (25)                        32'h00000000         32'h00000000
  (26)                        32'h00000000         32'h00000000
                    Now       800 ns      ns  100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns
               Cursor 1       0 ns        0 ns
```

Fig. 3. Data memory of the MIPS processor for the first program. Note that Mem[8-11], which corresponds to the 2nd memory register value, has 19 by the final clock cycle (clock cycle 8).

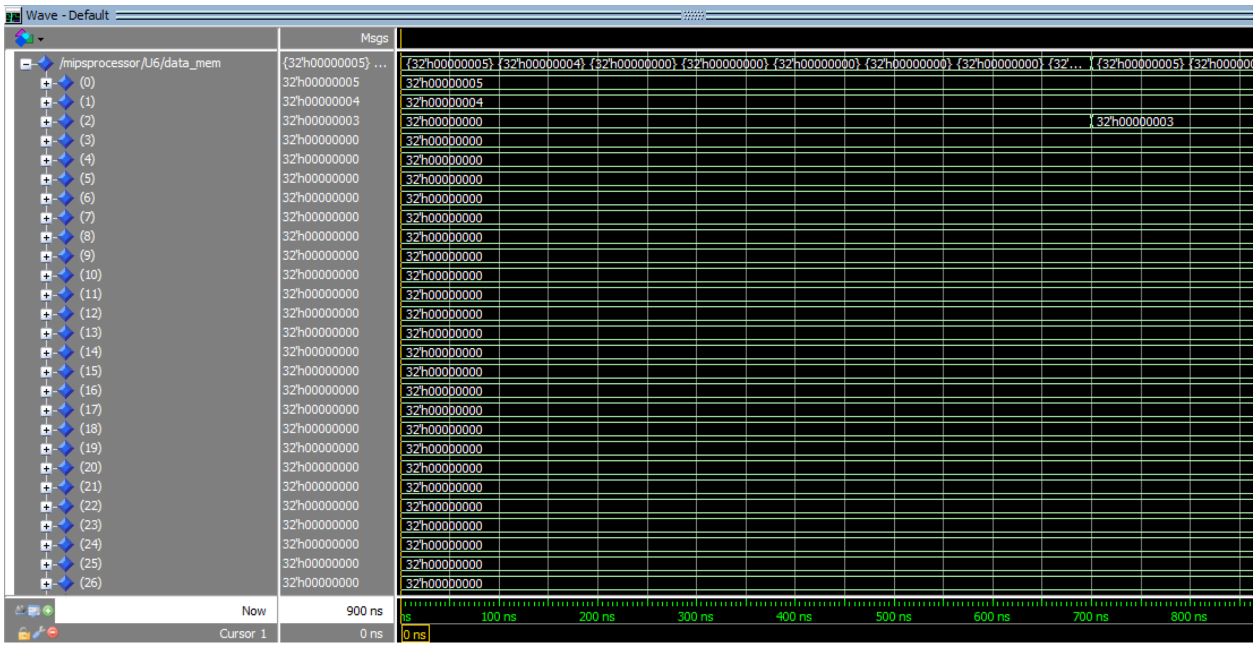
There is no special functionality for this first component, and the pipeline works without difficulty. Notice that there are 2 updated values in clock cycle 8 (700 ns – 800 ns) since the sub instruction writes to the registers (and does so in the WB phase) while the sw instruction writes to data memory in the MEM phase (which occurs one clock cycle earlier than the WB phase). Other than that, every write occurs one clock cycle after another since all instructions besides sw store in the registers during the WB phase.

The second problem illustrated the issue with the current implementation of the pipeline (which will be discussed further in the "Further Enhancements…" section) since this pipeline does not resolve hazards. The second program featured a data hazard (specifically a RAW dependency) between the final 2 instructions with register $s7. Given the design of the pipeline, the sw instruction would fetch the wrong value from registers and would then ultimately store the wrong value in memory. Figs. 4 and 5 show the results of the registers and the data memory, respectively.



```
/mipsprocessor/U3/regs  {32'h00000000} ...  {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} ... {32'h000000... {32'h000000... {32'h000000... {32'h00000000} {32'h00000
  (0)    32'h00000000   32'h00000000
  (1)    32'h00000000   32'h00000000
  (2)    32'h00000000   32'h00000000
  (3)    32'h00000000   32'h00000000
  (4)    32'h00000000   32'h00000000
  (5)    32'h00000000   32'h00000000
  (6)    32'h00000000   32'h00000000
  (7)    32'h00000000   32'h00000000
  (8)    32'h00000000   32'h00000000
  (9)    32'h00000004   32'h00000004
  (10)   32'h00000004   32'h00000004
  (11)   32'h00000000   32'h00000000
  (12)   32'h00000000   32'h00000000
  (13)   32'h00000000   32'h00000000
  (14)   32'h00000000   32'h00000000
  (15)   32'h00000000   32'h00000000
  (16)   32'h00000005   32'h00000000   32'h00000005
  (17)   32'h00000004   32'h00000000   32'h00000004
  (18)   32'h00000000   32'h00000000
  (19)   32'h00000013   32'h00000000   32'h00000013
  (20)   32'h0000000E   32'h0000000E
  (21)   32'h00000005   32'h00000005
  (22)   32'h00000008   32'h00000008
  (23)   32'h00000006   32'h00000003   32'h00000006
  (24)   32'h00000000   32'h00000000
  (25)   32'h00000000   32'h00000000
  (26)   32'h00000000   32'h00000000
Now        900 ns
Cursor 1     0 ns   0 ns
```

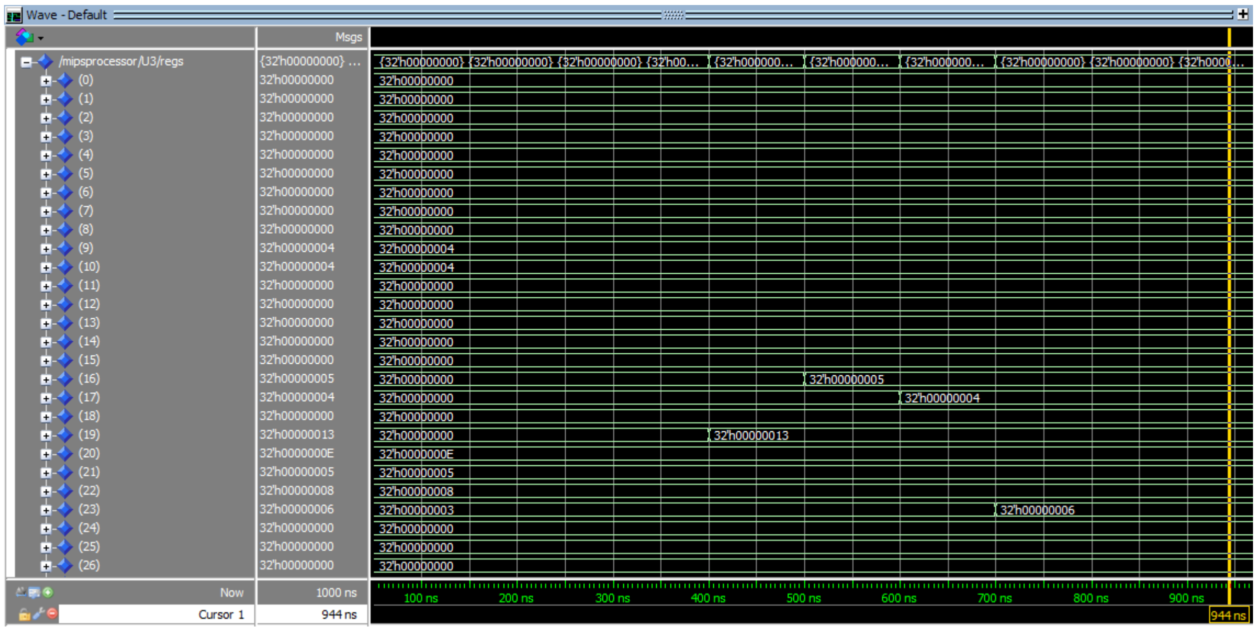
Fig. 4. Resulting registers for program 2. Notice that $s3 has 19, $s0 has 5, $s1 has 4, and $s7 has 6 by clock cycle 8. All these values are expected given the design and correct, even without forwarding.


```
Wave - Default
                                          Msgs
/mipsprocessor/U6/data_mem   {32'h00000005} ...
  (0)                        32'h00000005
  (1)                        32'h00000004
  (2)                        32'h00000003
  (3)                        32'h00000000
  (4)                        32'h00000000
  (5)                        32'h00000000
  (6)                        32'h00000000
  (7)                        32'h00000000
  (8)                        32'h00000000
  (9)                        32'h00000000
  (10)                       32'h00000000
  (11)                       32'h00000000
  (12)                       32'h00000000
  (13)                       32'h00000000
  (14)                       32'h00000000
  (15)                       32'h00000000
  (16)                       32'h00000000
  (17)                       32'h00000000
  (18)                       32'h00000000
  (19)                       32'h00000000
  (20)                       32'h00000000
  (21)                       32'h00000000
  (22)                       32'h00000000
  (23)                       32'h00000000
  (24)                       32'h00000000
  (25)                       32'h00000000
  (26)                       32'h00000000
                       Now   900 ns
                  Cursor 1   0 ns
```

{32'h00000005} {32'h00000004} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'… {32'h00000005} {32'h00000
32'h00000005
32'h00000004
32'h00000000
32'h00000003
ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
0 ns

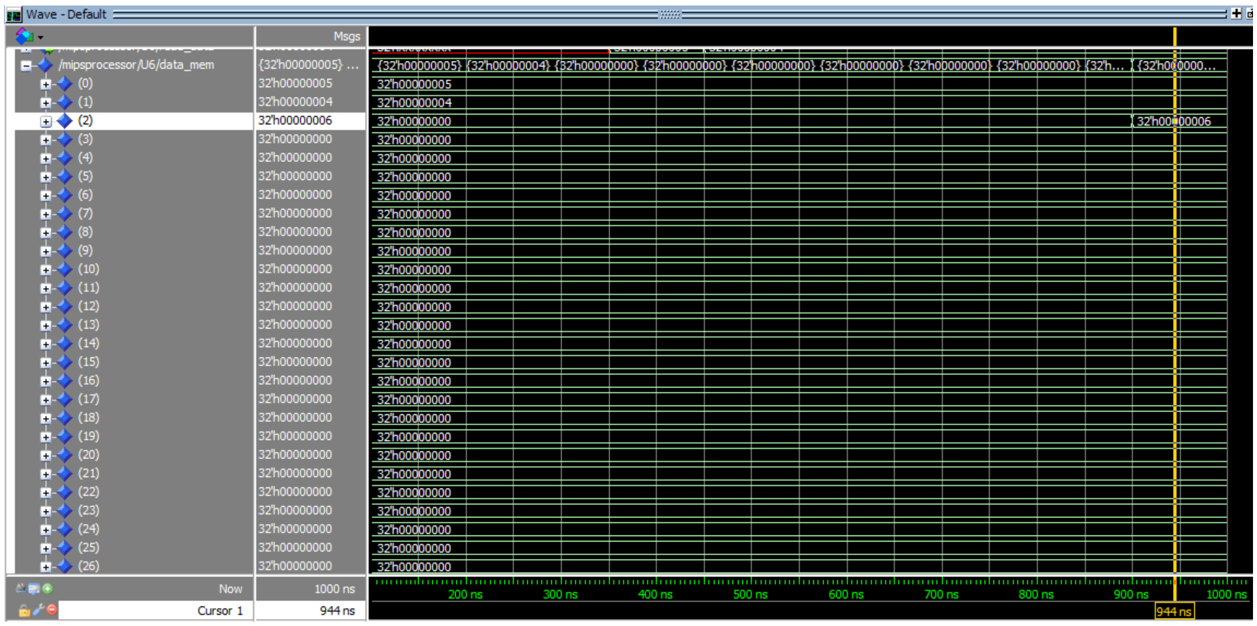
Fig. 5. Resulting data memory for program 2. Notice that M[8-11] (the 2nd memory location) has the result of 3 in clock cycle 8. This is not the correct value, but it is expected given the design and input instruction code.

It is desirable to fix the issue observed in program 2, but without forwarding, it becomes necessary to insert stalls (2 noop instructions) in order to allow the sub instruction to update $s7 with enough time. In order to allow for stalls to be inserted, the dependent instructions were buffered with the code “FFFFFFFF” since a binary opcode of “111111” does not correspond with any value and thus sets all control flags to ‘0’. The result of this change is shown through the registers and data memory in Figs. 6 and 7 on the following page.



```
/mipsprocessor/U3/regs  {32'h00000000} ...  {32'h00000000} {32'h00000000} {32'h00000000} {32'h00... {32'h000000... {32'h000000... {32'h000000... {32'h00000000} {32'h00000000} {32'h0000...
  (0)   32'h00000000   32'h00000000
  (1)   32'h00000000   32'h00000000
  (2)   32'h00000000   32'h00000000
  (3)   32'h00000000   32'h00000000
  (4)   32'h00000000   32'h00000000
  (5)   32'h00000000   32'h00000000
  (6)   32'h00000000   32'h00000000
  (7)   32'h00000000   32'h00000000
  (8)   32'h00000000   32'h00000000
  (9)   32'h00000004   32'h00000004
  (10)  32'h00000004   32'h00000004
  (11)  32'h00000000   32'h00000000
  (12)  32'h00000000   32'h00000000
  (13)  32'h00000000   32'h00000000
  (14)  32'h00000000   32'h00000000
  (15)  32'h00000000   32'h00000000
  (16)  32'h00000005   32'h00000000   32'h00000005
  (17)  32'h00000004   32'h00000000   32'h00000004
  (18)  32'h00000000   32'h00000000
  (19)  32'h00000013   32'h00000000   32'h00000013
  (20)  32'h0000000E   32'h0000000E
  (21)  32'h00000005   32'h00000005
  (22)  32'h00000008   32'h00000008
  (23)  32'h00000006   32'h00000003   32'h00000006
  (24)  32'h00000000   32'h00000000
  (25)  32'h00000000   32'h00000000
  (26)  32'h00000000   32'h00000000
Now       1000 ns
Cursor 1  944 ns
```

Fig. 6. Resulting registers for Program 3. Notice that the values of the registers are unchanged from Program 2 (as seen in Fig. 4) since the only change is an insertion of stalls after the register instructions.

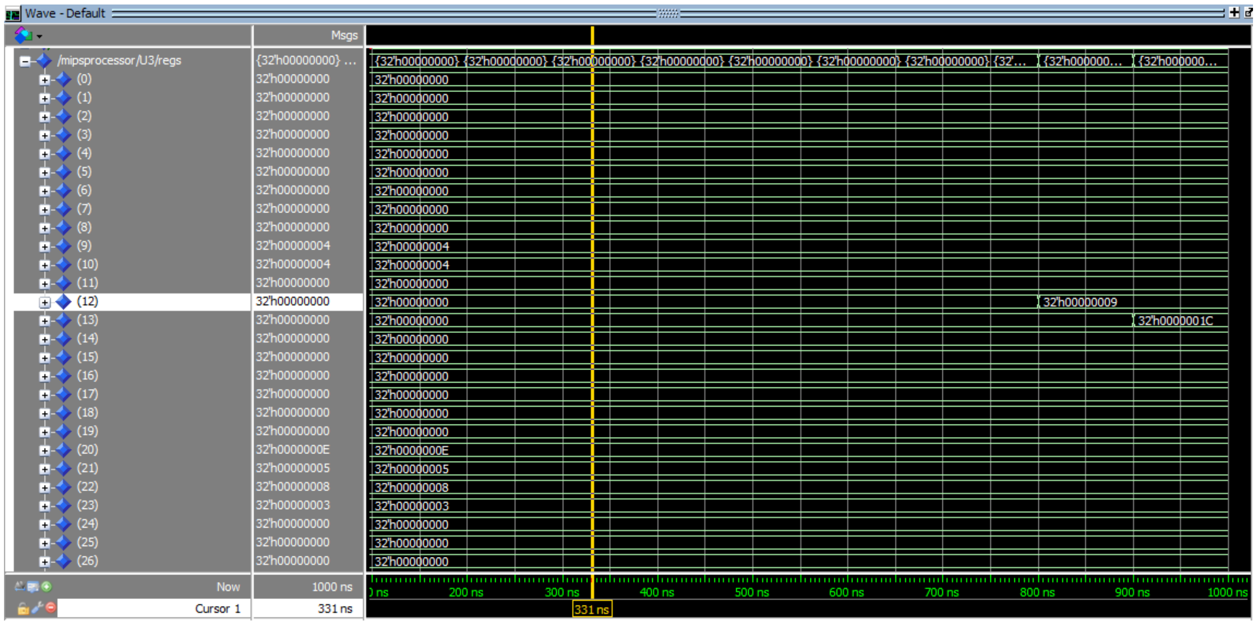


```
/mipsprocessor/U6/data_mem  {32'h00000005} ...  {32'h00000005} {32'h00000004} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h... {32'h000000...
  (0)   32'h00000005   32'h00000005
  (1)   32'h00000004   32'h00000004
  (2)   32'h00000006   32'h00000000   32'h00000006
  (3)   32'h00000000   32'h00000000
  (4)   32'h00000000   32'h00000000
  (5)   32'h00000000   32'h00000000
  (6)   32'h00000000   32'h00000000
  (7)   32'h00000000   32'h00000000
  (8)   32'h00000000   32'h00000000
  (9)   32'h00000000   32'h00000000
  (10)  32'h00000000   32'h00000000
  (11)  32'h00000000   32'h00000000
  (12)  32'h00000000   32'h00000000
  (13)  32'h00000000   32'h00000000
  (14)  32'h00000000   32'h00000000
  (15)  32'h00000000   32'h00000000
  (16)  32'h00000000   32'h00000000
  (17)  32'h00000000   32'h00000000
  (18)  32'h00000000   32'h00000000
  (19)  32'h00000000   32'h00000000
  (20)  32'h00000000   32'h00000000
  (21)  32'h00000000   32'h00000000
  (22)  32'h00000000   32'h00000000
  (23)  32'h00000000   32'h00000000
  (24)  32'h00000000   32'h00000000
  (25)  32'h00000000   32'h00000000
  (26)  32'h00000000   32'h00000000
Now       1000 ns
Cursor 1  944 ns
```

Fig. 7. Resulting data memory for Program 3. Notice that the 2nd memory value (which corresponds to M[8-11]) now updates later than it did in Program 2, but the value is now correct.

The results with the stall are now correct, which indicates that the processor is working fully as expected for non-branch instructions. These instructions are tested in Program 4.

In Program 4, there is once again a hazard (however it is a branch hazard in this portion). This is resolved by inserting three pipeline stalls. Since the stalls were shown to work in Program 3, this portion illustrates the functionality of the branch instructions (in this example, beq). The resulting registers can be viewed in Fig. 8. Note that there are no memory accesses, so for clarity the data memory has been omitted.

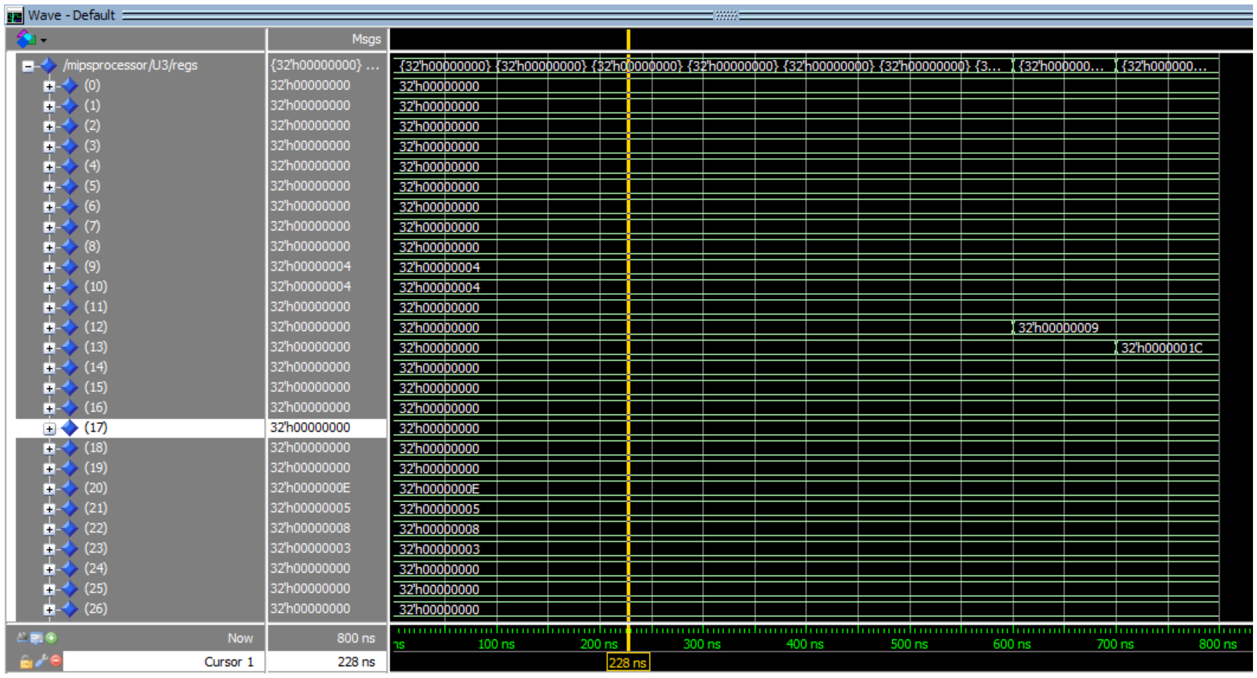


	Msgs	
/mipsprocessor/U3/regs	{32'h00000000} ...	{32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'... {32'h000000... {32'h000000...
(0)	32'h00000000	32'h00000000
(1)	32'h00000000	32'h00000000
(2)	32'h00000000	32'h00000000
(3)	32'h00000000	32'h00000000
(4)	32'h00000000	32'h00000000
(5)	32'h00000000	32'h00000000
(6)	32'h00000000	32'h00000000
(7)	32'h00000000	32'h00000000
(8)	32'h00000000	32'h00000000
(9)	32'h00000004	32'h00000004
(10)	32'h00000004	32'h00000004
(11)	32'h00000000	32'h00000000
(12)	32'h00000000	32'h00000000 32'h00000009
(13)	32'h00000000	32'h00000000 32'h0000001C
(14)	32'h00000000	32'h00000000
(15)	32'h00000000	32'h00000000
(16)	32'h00000000	32'h00000000
(17)	32'h00000000	32'h00000000
(18)	32'h00000000	32'h00000000
(19)	32'h00000000	32'h00000000
(20)	32'h0000000E	32'h0000000E
(21)	32'h00000005	32'h00000005
(22)	32'h00000008	32'h00000008
(23)	32'h00000003	32'h00000003
(24)	32'h00000000	32'h00000000
(25)	32'h00000000	32'h00000000
(26)	32'h00000000	32'h00000000

Fig. 8. The resulting registers from Program 4. Notice that register \$t4 has 9 in cycle 9 and register \$t5 has 28 in clock cycle 10. These are the expected and desired results.

The results from Program 4 show that the branch instruction is resolved in the EX stage and ultimately the value is fed back to the two mux modules which select from the incremented PC value, the branch PC value and the jump PC value. In this example, the branch PC value is selected so that the non-executed instructions are skipped and the branch instructions resolve at the correct clock cycle.

The final two portions of this experiment (Problems 5 and 6) explored the requirements of resolving a branch in the ID phase to reduce the total number of stall cycles for a branch instruction. In order to accomplish this, a new module had to be developed which would simply compare the values of the two registers for beq and bne instructions. This is possible in the MIPS processor since the comparison is quite simple. A bitwise XOR and an OR of the solution bits shows whether the registers are the same (a result of 1 implies that they differ and 0 implies they are the same). In VHDL, the implementation is even simpler since each register value is stored as a STD_LOGIC_VECTOR. These may be tested inline for equality, which simplified the implementation. Finally, the result of the branch is already able to be calculated since the PC incremented value and the immediate are available in the ID stage. Thus, a simple rewiring was sufficient for the processor to work again. This reduced the stall cycles to 1, which can be seen in Program 5 results in Fig. 9.

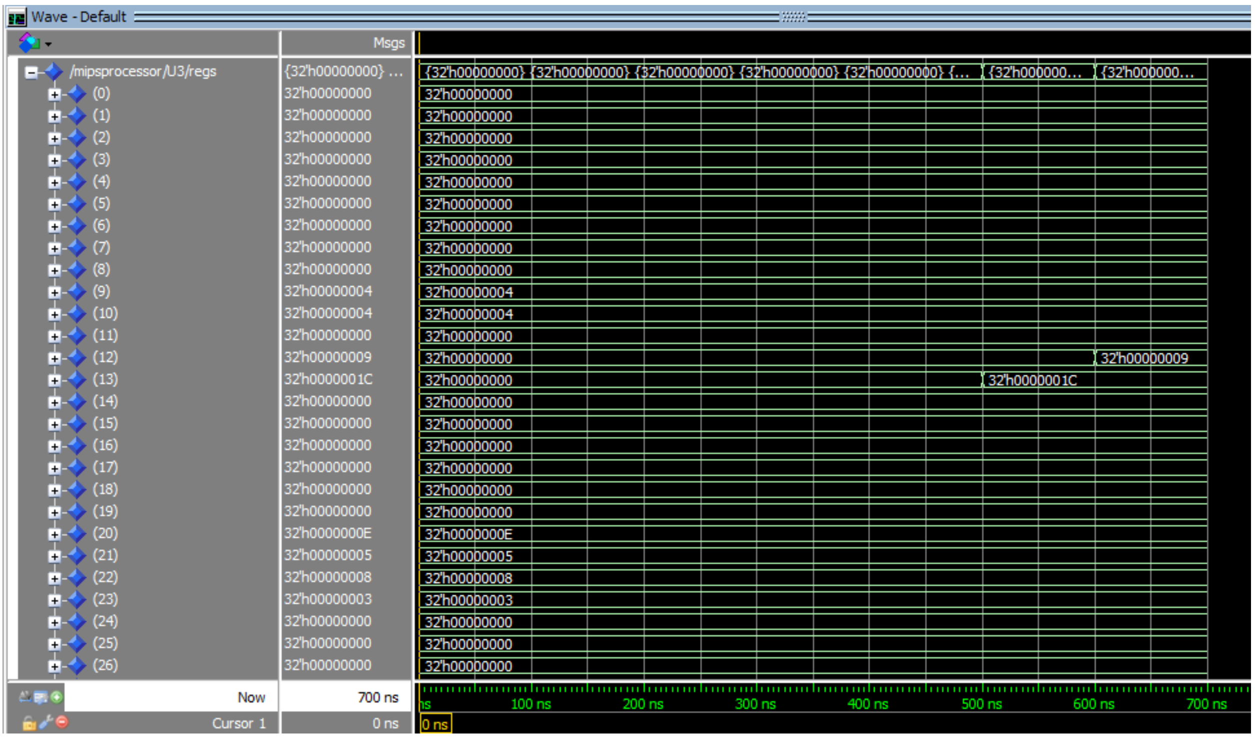


```
Wave - Default
                          Msgs
/mipsprocessor/U3/regs    {32'h00000000} ...   {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {3... {32'h000000... {32'h000000...
  (0)                     32'h00000000         32'h00000000
  (1)                     32'h00000000         32'h00000000
  (2)                     32'h00000000         32'h00000000
  (3)                     32'h00000000         32'h00000000
  (4)                     32'h00000000         32'h00000000
  (5)                     32'h00000000         32'h00000000
  (6)                     32'h00000000         32'h00000000
  (7)                     32'h00000000         32'h00000000
  (8)                     32'h00000000         32'h00000000
  (9)                     32'h00000004         32'h00000004
  (10)                    32'h00000004         32'h00000004
  (11)                    32'h00000000         32'h00000000
  (12)                    32'h00000000         32'h00000000                                   32'h00000009
  (13)                    32'h00000000         32'h00000000                                                  32'h0000001C
  (14)                    32'h00000000         32'h00000000
  (15)                    32'h00000000         32'h00000000
  (16)                    32'h00000000         32'h00000000
  (17)                    32'h00000000         32'h00000000
  (18)                    32'h00000000         32'h00000000
  (19)                    32'h00000000         32'h00000000
  (20)                    32'h0000000E         32'h0000000E
  (21)                    32'h00000005         32'h00000005
  (22)                    32'h00000008         32'h00000008
  (23)                    32'h00000003         32'h00000003
  (24)                    32'h00000000         32'h00000000
  (25)                    32'h00000000         32'h00000000
  (26)                    32'h00000000         32'h00000000
                    Now   800 ns    ns  100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns
               Cursor 1   228 ns               228 ns
```

Fig. 9. Resulting registers from Program 5. Notice that register $t4 has 9 in cycle 7 and register $t5 has 28 in cycle 8. This is the desired result since the instruction affecting register $t3 is skipped.

The delay from a branch instruction can be effectively eliminated by taking advantage of the branch delay slot. By placing an instruction which should always execute in the branch delay, the stall can be avoided. Noticing that the final add instruction (which stores in register

$t5) will always execute allows the program to be rewritten to place it below the beq instruction so that there is no delay. The result of this program, Program 6, is shown in Fig. 10.



```
Wave - Default
                                   Msgs
/mipsprocessor/U3/regs   {32'h00000000} ...   {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {32'h00000000} {... {32'h000000... {32'h000000...
  (0)                    32'h00000000         32'h00000000
  (1)                    32'h00000000         32'h00000000
  (2)                    32'h00000000         32'h00000000
  (3)                    32'h00000000         32'h00000000
  (4)                    32'h00000000         32'h00000000
  (5)                    32'h00000000         32'h00000000
  (6)                    32'h00000000         32'h00000000
  (7)                    32'h00000000         32'h00000000
  (8)                    32'h00000000         32'h00000000
  (9)                    32'h00000004         32'h00000004
  (10)                   32'h00000004         32'h00000004
  (11)                   32'h00000000         32'h00000000
  (12)                   32'h00000009         32'h00000000                                                                32'h00000009
  (13)                   32'h0000001C         32'h00000000                                                 32'h0000001C
  (14)                   32'h00000000         32'h00000000
  (15)                   32'h00000000         32'h00000000
  (16)                   32'h00000000         32'h00000000
  (17)                   32'h00000000         32'h00000000
  (18)                   32'h00000000         32'h00000000
  (19)                   32'h00000000         32'h00000000
  (20)                   32'h0000000E         32'h0000000E
  (21)                   32'h00000005         32'h00000005
  (22)                   32'h00000008         32'h00000008
  (23)                   32'h00000003         32'h00000003
  (24)                   32'h00000000         32'h00000000
  (25)                   32'h00000000         32'h00000000
  (26)                   32'h00000000         32'h00000000
                    Now  700 ns               ns   100 ns   200 ns   300 ns   400 ns   500 ns   600 ns   700 ns
               Cursor 1  0 ns                 0 ns
```

Fig. 10. The resulting registers from Program 6. Notice that register $t5 is now written before register $t4 and $t5 is set to 28 in clock cycle 6 and $t4 is set to 9 in clock cycle 7. These results are the expected and the correct results.

Further Enhancements and Future Work

Unfortunately, even by resolving the branch statement in the ID stage, the effective CPI of this pipeline is far from 1. This is because there are still data hazards between statements and, in the worst case, the branch delay slot cannot be leveraged to reduce stall cycles. It is desirable to add a module which can detect these hazards in order to reduce them and to forward data and bypass delays. The future work related to this project will center exactly on this. Additionally, in development of this processor, it was decided that it would be valuable to support instructions not explicitly required for the task at hand. Thus, the ALU was enhanced to accept shift instructions and all logical instructions (e.g. XOR, OR, etc.). Also, a jump module was added to support j-type instructions. In the future, these functions will be fully tested in the pipeline in order to ensure that the processor is as close to the actual functionality as possible.

Conclusory Statements

This project largely centered on the effects of the buffers between each phase of the pipeline. Interestingly, the development of these buffers was quite simple and the wiring was the most difficult component. Ensuring that everything was passed as needed was somewhat complicated, but especially illuminating. The overhead hardware required in order to support the pipeline illustrates why single-cycle was likely developed first (in addition to the relative simplicity of single-cycle). Additionally, the insertion of even more hardware to support early branch resolution demonstrated the tradeoffs between speed and cost of hardware (and even more so, size complexity) in a very tangible way. One of the interesting difficulties which arose was the allowing of a read to occur after the write of a register. This allowed to reduce the total number of stalls as well, but required a small amount of additional code that was not immediately apparent at the outset. Ultimately, this project was quite illuminating and demonstrated the value of the pipeline as a whole.